1. Шейкерная сортировка.

Шейкерная сортировка или **сортировка перемешиванием** — усовершенствованная разновидность сортировки пузырьком, при которой сортировка производится в двух направлениях, меняя направление при каждом проходе.

Проанализировав алгоритм пузырьковой сортировки, можно заметить:

- если при обходе части массива не происходит обмен элементов, то эту часть можно исключить, так как она уже отсортирована;
- при проходе от конца массива к началу минимальное значение сдвигается на первую позицию, при этом максимальный элемент перемещается только на один индекс вправо.

Исходя из приведенных идей, модифицируем сортировку пузырьком следующим образом:

- на каждой итерации фиксируем границы части массива, в которой происходит обмен;
- массив просматривается поочередно от начала массива к концу и от конца к началу;

Мы перемещаем минимальный элемент в начало массива, а максимальный — в конец, после этого выбрасываем индексы этих элементов из рабочей области. Таким образом, за каждую итерацию просматриваемая часть массива уменьшается на два элемента.

Фрагмент кода (сортировка по возрастанию массива a размера n):

```
{Шейкерная сортировка}

  firstIndex := 1;          {индекс первого элемента рабочей зоны массива}
  lastIndex := n; {индекс последнего элемента рабочей зоны массива}

  while firstIndex < lastIndex do
  begin
    {проход слева направо}
    for i:= firstIndex to lastIndex-1 do
      if a[i] > a[i+1] then
      begin
        {обмен элементов}
        swap(a[i],a[i+1]);
      end;

    {проход справа налево}
    for i:= lastIndex downto firstIndex+1 do
      if a[i] < a[i-1] then
      begin
        {обмен элементов}
        swap(a[i],a[i-1]);
      end;

    {уменьшаем просматриваемую область}
    firstIndex := firstIndex + 1;
    lastIndex := lastIndex - 1;
     end;
```

2. Сортировка бинарными вставками.

Сортировка бинарными вставками является улучшением сортировки простыми вставками: поиск "подходящего места" в упорядоченной последовательности можно вести более экономичным способом, который называется *Двоичный поиск в упорядоченной последовательности*. Он напоминает детскую игру "больше-меньше": после каждого сравнения обрабатываемая последовательность сокращается в два раза.

Пусть, к примеру, нужно найти место для элемента **7** в таком массиве:

[2 4 6 8 10 12 14 16 18]

Найдем средний элемент этой последовательности (**10**) и сравним с ним семерку. После этого все, что больше **10** (да и саму десятку тоже), можно смело исключить из дальнейшего рассмотрения:

[2 4 6 8] 10 12 14 16 18

Снова возьмем середину в отмеченном куске последовательности, чтобы сравнить ее с семеркой.

Однако здесь нас поджидает небольшая проблема: точной середины у новой последовательности нет, поэтому нужно решить, который из двух центральных элементов станет этой "серединой". От того, к какому краю будет смещаться выбор в таких "симметричных" случаях, зависит окончательная реализация нашего алгоритма.

Новой "серединой" последовательности всегда будет становиться левый центральный элемент. Это соответствует вычислению номера "середины" по формуле

nomer_sred:= (nomer_lev + nomer_prav) div 2

Итак, отсечем половину последовательности:

2 4 [6 8] 10 12 14 16 18

И снова:

2 4 6 [8] 10 12 14 16 18

2 4 6] [8 10 12 14 16 18

Таким образом, мы нашли в исходной последовательности место, "подходящее" для нового элемента.

Если бы в той же самой последовательности нужно было найти позицию не для семерки, а для девятки, то последовательность границ рассматриваемых промежутков была бы такой:

[2 4 6 8] 10 12 14 16 18

2 4 [6 8] 10 12 14 16 18

2 4 6 [8] 10 12 14 16 18

2 4 6 8] [10 12 14 16 18

Из приведенных примеров уже видно, что поиск ведется до тех пор, пока левая граница не окажется правее **(!)** правой границы.

Кроме того, по завершении этого поиска последней левой границей окажется как раз тот элемент, на котором необходимо закончить сдвиг "хвоста" последовательности.

Фрагмент кода (сортировка по возрастанию массива a размера n):

```
{сортировка бинарными вставками}
for i:=2 to n do
  begin
   x:=a[i];{запомним элемент}
   l:=1;{левый край}
   r:=i-1;{правый}
   while l<=r do {пока левый не больше правого}
    begin
     m:=(l+r) div 2;{находим середину}
     if x<a[m] then r:=m-1{если элемент меньше среднего,
                           правый край левее середины}
     else l:=m+1{иначе левый правее середины}
    end;
   for j:=i-1 downto l do a[j+1]:=a[j];{сдвигаем массив вправо на 1}
   a[l]:=x{вставляем элемент на место}
  end; { Окончание алгоритма сортировки}
```

3. Быстрая сортировка

Отличительной особенностью быстрой сортировки является операция разбиения массива на две части относительно опорного элемента. Например, если последовательность требуется упорядочить по возрастанию, то в левую часть будут помещены все элементы, значения которых меньше значения опорного элемента, а в правую элементы, чьи значения больше или равны опорному.

Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов. Если длина какой-то из получившихся в результате разбиения частей превышает один элемент, то для нее нужно рекурсивно выполнить упорядочивание, т. е. повторно запустить алгоритм на каждом из отрезков.

Таким образом, алгоритм быстрой сортировки включает в себя два основных этапа:

- разбиение массива относительно опорного элемента;
- рекурсивная сортировка каждой части массива.

Общая схема разбиения массива (сортировка по возрастанию):

1. вводятся указатели *first* и *last* для обозначения начального и конечного элементов последовательности, а также опорный элемент *mid*;
2. вычисляется значение опорного элемента (*first*+*last*)/2, и заносится в переменную *mid*;

3. указатель *first* смещается с шагом в 1 элемент к концу массива до тех пор, пока *mas*[*first*]>*mid*. А указатель *last* смещается от конца массива к его началу, пока *mas*[*last*]<*mid*;
4. каждые два найденных элемента меняются местами;
5. пункты 3 и 4 выполняются до тех пор, пока first<last.

После разбиения последовательности следует проверить условие на необходимость дальнейшего продолжения сортировки его частей. Этот этап будет рассмотрен позже, а сейчас на конкретном примере выполним разбиение массива.

Имеется массив целых чисел *mas*, состоящий из 8 элементов. Начальным значением *first* будет 1, а *last* – 8. Пройденная часть закрашивается голубым цветом.



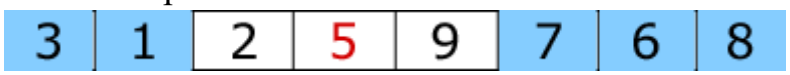
В качестве опорного элемента возьмем элемент со значением 5, и индексом 4. Его мы вычислили, используя выражение (*first*+*last*)/2, отбросив дробную часть. Теперь *mid*=5.



Первый элемент левой части сравнивается с *mid*. *mas*[1]>*mid*, следовательно *first* остается равным 1. Далее, элементы правой части сравниваются с *mid*. Проверяется элемент с индексом 8 и значением 8. *mas*[8]>*mid*, следовательно *last* смещается на одну позицию влево. *mas*[7]<*mid*, следовательно *last* остается равным 7. На данный момент *first*=1, а *last*=7. Первый и седьмой элементы меняются местами. Оба указателя смещаются на одну позицию каждый в своем направлении.



Алгоритм снова переходит к сравнению элементов. Второй элемент сравнивается с опорным: *mas*[2]>mid, следовательно *first* остается равным 2. Далее, элементы правой части сравниваются с *mid*. Проверяется элемент с индексом 6 и значением 1: *mas*[6]<*mid*, следовательно *last* не изменяет своей позиции. На данный момент *first*=2, а *last*=6. Второй и шестой элементы меняются местами. Оба указателя смещаются на одну позицию каждый в своем направлении.



Алгоритм снова переходит к сравнению элементов. Третий элемент сравнивается с опорным: *mas*[3]<*mid*, следовательно *first* смещается на одну позицию вправо. Далее, элементы правой части сравниваются с *mid*. Проверяется элемент с индексом 5 и значением 9: *mas*[5]>*mid*,

следовательно *last* смещается на одну позицию влево. Теперь *first*=*last*=4, а значит, условие *first*<*last* не выполняется, этап разбиения завершается.



На этом этап разбиения закончен. Массив разделен на две части относительно опорного элемента. Осталось произвести рекурсивное упорядочивание его частей.

Рекурсивное доупорядочивание

Если в какой-то из получившихся в результате разбиения массива частей находится больше одного элемента, то следует произвести рекурсивное упорядочивание этой части, то есть выполнить над ней операцию разбиения, описанную выше. Для проверки условия «количество элементов > 1», нужно действовать примерно по следующей схеме:

Имеется массив *mas*[*L*..*R*], где *L* и *R* – индексы крайних элементов этого массива. По окончанию разбиения, указатели *first* и *last* оказались примерно в середине последовательности, тем самым образуя два отрезка: левый от *L* до *last* и правый от *first* до *R*. Выполнить рекурсивное упорядочивание левой части нужно в том случае, если выполняется условие *L*<*last*. Для правой части условие аналогично: *first*<*R*.

Фрагмент кода (сортировка по возрастанию массива a размера n):

```
{процедура сортировки}
procedure quicksort(var mas: arr; first, last: integer);
var f, l, mid, count: integer;
begin
f:=first;
l:=last;
mid:=mas[(f+l) div 2]; {вычисление опорного элемента}
repeat
      while mas[f]<mid do inc(f);
      while mas[l]>mid do dec(l);
      if f<=l then {перестановка элементов}
      begin
            count:=mas[f];
            mas[f]:=mas[l];
            mas[l]:=count;
            inc(f);
            dec(l);
      end;
until f>l;
if first<l then quicksort(A, first, l);
if f<last then quicksort(A, f, last);
end;
```

Вызов процедуры:

```
first:=1; last:=n;
quicksort(a, first, last);
```

Задание на пару: Отсортировать элементы массива по возрастанию и по убыванию, используя каждую из предложенных выше сортировок.

Домашнее задание: Подготовиться к контрольной работе.